МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Уральский федеральный университет имени
первого Президента России Б. Н. Ельцина»

ПОСТРОЕНИЕ ТИПОВЫХ МОДЕЛЕЙ АРПСС (ARIMA)

Методические указания к выполнению
практического задания № 4

Екатеринбург
2024

Содержание

Введение..3

1. Задание на лабораторную работу 3

2. Требования к оформлению отчета..16

Уральский федеральный университет имени первого Президента России Б.Н.Ельцина

Введение

Напомним, что в общем виде модель авторегрессии – скользящего среднего порядка (p, q) АРПСС (ARIMA) выглядит как:

$$\tilde{z}_t = \phi_1 \tilde{z}_{t-1} + \phi_2 \tilde{z}_{t-2} + \ldots + \phi_p \tilde{z}_{t-p} + a_t - \theta_1 a_{t-1} - \theta_2 a_{t-2} - \ldots - \theta_q a_{t-q}.$$

Эта модель временных рядов имеет целый ряд преимуществ в сравнении с другими моделями, одно из которых – это возможность их оперативного прогноза по построенной модели. В связи с этим ни одна методика анализа и изучения временных рядов не может обойтись без рассмотрения подобного класса задач.

1. Задание на лабораторную работу

Результатом выполнения лабораторной работы является оформленный отчет в виде *Jupyter*-тетради, в котором должны быть представлены и отражены все нижеперечисленные пункты:

1) Сначала импортируйте в свой код нужные библиотеки, функции и т.д.

```
import numpy as np
import numpy.random as rand
import matplotlib.pyplot as plt
import h5py
from statsmodels.tsa import api as tsa
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.tsa.arima.model import ARIMA
```

2) Для начала попробуем создать собственные АРПСС ряды первого и второго порядков и изучить их автокорреляционные функции.
3) Создадим два АР(1) процесса первого порядка:

$$z_t = 0.8 z_{t-1} + a_t \quad \text{и} \quad z_t = -0.8 z_{t-1} + a_t$$

где a_t – случайная нормально распределенная величина малой амплитуды (порядка 0.2), $z_0 = 1$.

```
z1 = np.zeros(100)
z2 = np.zeros(100)
z1[0] = 1
z2[0] = 1
for i in range(1,100):
   z1[i] = 0.8 * z1[i - 1] + 0.2 * np.random.randn()
   z2[i] = -0.8 * z2[i - 1] + 0.2 * np.random.randn()
plt.figure(figsize = (10, 5))
plt.plot(z1, 'b')
plt.plot(z2, 'r')
plt.show()
```

4) Постройте для этих рядов функции автокорреляции с помощью функции **plot_acf**:

```
plt.figure(figsize = (10, 5))
plot_acf(z1, lags=50)
plot_acf(z2, lags=50)
plt.show()
```

5) Сравните эти графики между собой: укажите их сходства и различия, а также характерные особенности, которые позволяют отнести их к модели АР первого порядка.

Уральский федеральный университет имени первого Президента России Б.Н.Ельцина

6) Аналогичным образом постройте два СС(1) процесса среднего-скользящего первого порядка:

$$z_t = a_t - 0.8a_{t-1} \quad \text{и} \quad z_t = a_t - (-0.8)a_{t-1}$$

где a_t – случайная нормально распределенная величина

```
z3 = np.zeros(100)
z4 = np.zeros(100)
ar = 0.2 * np.random.randn(100)
for i in range(1, 100):
   z3[i] = ar[i] - 0.8 * ar[i - 1]
   z4[i] = ar[i] + 0.8 * ar[i - 1]
plt.figure(figsize = (10, 5))
plt.plot(z3, 'b')
plt.plot(z4, 'r')
plt.show()
```

7) Постройте для этих рядов функции автокорреляции подобно п. 4 выше, достаточно взять 20 лагов (lags=20).
8) Сравните эти графики между собой: укажите их сходства и различия, а также характерные особенности, которые позволяют отнести их к модели СС первого порядка.
9) Удостоверьтесь, что для модели СС(1) коэффициенты автокорреляции приблизительно соответствуют формуле

$$\rho_k = \begin{cases} \dfrac{-\theta_1}{1+\theta_1^2}, & k = 1 \\ 0, & k \geq 2 \end{cases}$$

где *p* – коэффициенты АКФ, θ – параметр модели СС(1)

10) Наконец, создайте временной ряд процесса АРСС(1, 1):

$z_t = 0.8z_{t-1} + a_t - 0.3a_{t-1}$ и $z_t = -0.8z_{t-1} + a_t - 0.3a_{t-1}$

где a_t – случайная нормально распределенная величина, $z_0 = 1$.

Напишите код *Python* самостоятельно на основе комбинации предыдущих примеров. Постройте графики этих рядов и графики их автокорреляционных функций.

11) Есть и другой, более высокоуровневый способ генерации рядов АРПСС. Используем следующую функцию для создания АРСС (2, 2):

```
from statsmodels.tsa.arima_process import arma_generate_sample
ar = np.array([0.75, -0.25])    # задаем коэффициенты АР
ma = np.array([0.65, 0.35])    # задаем коэффициенты СС
y = arma_generate_sample(np.r_[1, -ar], np.r_[1, ma], 100)
# создаем ВР для АРСС (2, 2) = АРПСС (2, 0, 2) из 100 отсчетов
```

Для получившегося ВР постройте его график и изображение АКФ. Укажите характерные особенности получившейся АКФ, позволяющие охарактеризовать ее в принадлежности к классу моделей АРСС некоторого порядка.

12) Теперь проведем анализ неизвестного ряда на типовом примере, а затем каждый из студентов проводит анализ собственного ВР по вариантам (номер варианта = последние две цифры студенческого билета).

13) Начнем с общего тестового примера для всех. Значения исходного ряда (всего их 24) приведены ниже:

TEST = [0.00, 9.99, 12.89, 10.70, 5.12, -1.21, -6.50, -7.96, -4.30, 0.42, 3.41, 4.50, 3.57, 2.24, 1.78, 0.89, -1.20, -3.43, -2.35, -0.85, -0.21, -0.08, 0.95, 0.45]

14) Постройте график ВР и его автокорреляционную функцию.

По ним можно судить, что ВР, в достаточной степени, **стационарен**, а, так как, эта функция является **знакопеременной**, то один из членов АР модели имеет отрицательный вес.

15) Создадим три пробные модели АРПСС для проверки ряда на АР(1) = АРПСС(1, 0, 0), АР(2), АР(3):

```
arima1 = ARIMA(TEST, order = (1, 0, 0))          # создаем модель
model_fit1 = arima1.fit() # подгоняем под ВР
print(model_fit1.summary())        # выводим таблицу результатов
arima2 = ARIMA(TEST, order = (2, 0, 0))
model_fit2 = arima2.fit()
print(model_fit2.summary())
arima3 = ARIMA(TEST, order = (3, 0, 0))
model_fit3 = arima3.fit()
print(model_fit3.summary())
```

16) Будут выведены три таблицы со всевозможной информацией (зависит от версии библиотеки), например, как ниже:

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                   24
Model:                 ARIMA(2, 0, 0)   Log Likelihood                 -41.415
Date:                Tue, 12 Mar 2024   AIC                             90.831
Time:                        21:33:32   BIC                             95.543
Sample:                             0   HQIC                            92.081
                                 - 24
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.2885      0.644      0.448      0.654      -0.973       1.550
ar.L1          1.5046      0.051     29.407      0.000       1.404       1.605
ar.L2         -0.9639      0.023    -41.590      0.000      -1.009      -0.919
sigma2         1.4275      0.709      2.013      0.044       0.038       2.817
===================================================================================
Ljung-Box (L1) (Q):                   0.22   Jarque-Bera (JB):                 0.57
Prob(Q):                              0.64   Prob(JB):                         0.75
Heteroskedasticity (H):               0.54   Skew:                             0.09
Prob(H) (two-sided):                  0.40   Kurtosis:                         2.27
===================================================================================
```

17) В этой таблице значения 2 коэффициентов модели авторегрессии АР(2) написаны в виде **ar.L1** & **ar.L2** в столбце **coef**. Коэффициент **const** означает среднее значение ряда ошибок (по сути – мат. ожидание ряда остатков), а **sigma2** – дисперсия этого остатка (то есть СКВО в квадрате – отсюда и *2* в названии), и для нашей модели они не особо важны. Все коэффициенты модели оцениваются приближенно, поэтому у них есть погрешность, СКВО этой погрешности – в следующем столбце **std err**. В последующих столбцах – результаты проверки статистического *z*-теста, то есть проверка получившихся значений на значимость результатов. Прямой гипотезой является «случайность» коэффициентов (как если бы любое число подошло), альтернативной гипотезой – значимость их значений для модели. В данном случае **P>|z|** есть **p-value** равное 0, то есть прямая гипотеза отвергается. Тогда принимается альтернативная гипотеза о том, что наши коэффициенты **ar.L1** & **ar.L2** подобраны не случайно и значимы в данной модели.

18) Но у нас в результате получилось три модели = три таблицы. **Как по этим таблицам выбрать наилучшую модель?** Во-первых, стоит обратить внимание на значение **AIC** – информационный критерий Акаике, который показывает максимальное правдоподобие модели при штрафовании за избыточные параметры системы. Считается, что наилучшей будет модель с **наименьшим** значением критерия AIC.

19) Аналогично есть **BIC** – Байесовский информационный критерий, модификация AIC. Данный критерий налагает больший штраф на увеличение количества параметров по сравнению с AIC.

20) Аналогично есть **HQIC** –информационный критерий Ханнана-Куинна (Hannan-Quinn), который асимптотически более точный метод чем BIC для дискретных параметров.

21) В любом случае, **лучшей моделью будет та, что имеет наименьшее значение информационного критерия среди множества других**. Рекомендуется, в первую очередь, выбирать по критерию **BIC**, так как он сильнее штрафует за переобучение модели и увеличение числа параметров по сравнению с другими. В нашем случае для тестового ВР, для любых информационных критериев, это модель АР(2).

22) Другим методом выбора модели может служить построение моделей АРПСС выбранного порядка и с найденными коэффициентам на графиках совмещенно.

plt.plot(model_fit.fittedvalues)

Например, для приведенного примера модель АР(1) совсем слабо подходит к ВР, АР(2) и АР(3) близки, АР(3) почти не отличается от АР(2), но избыточен по числе параметров (3>2), а значит АР(2) является наиболее оптимальной моделью ВР.

23) В современных библиотеках *Python* есть и более продвинутые функции, автоматизирующие поиск наилучших простых ARMA моделей на основе информационного критерия **BIC**. Например, итоговый результат выбора модели из п.18-22 можно было получить всего в пару строчек кода:

```
import statsmodels.tsa.stattools as stt
stt.arma_order_select_ic(TEST)
```

В результате будут выведены значения информационного критерия **BIC** для моделей с комбинацией порядков АР до 4 и СС до 2. Ну а самое важное будет в последней строчке:

```
'bic_min_order': (2, 0)
```

То есть по критерию **BIC** наилучшей моделью будет ARMA (2, 0) или просто AR (2), то есть АР второго порядка, что мы и увидели из таблиц п. 16.

24) Теперь в зависимости от своего варианта, который определяется по последним двум цифрам студ. билета, выберите из выданных преподавателей ***mat*-файлов** тот, который имеет номер Вашего варианта и загрузите из него временной ряд ***Z***, например:

```
file = h5py.File('12.mat', 'r')
data = file.get('z12')
Z = np.array(data)
Z = Z.ravel()
```

25) Постройте график ВР и его автокорреляционную функцию.

26) Оцените порядок АРСС модели для данных Вашего варианта с помощью класса ARIMA. Для упрощения задачи выбора модели используйте только чистые АР или СС модели, то есть класс ARIMA с order = (p, 0, 0) или order = (0, 0, q).

27) Выберите модель с наиболее подходящей структурой и вычислите для нее коэффициенты.

Поясните в отчете выбор модели.

28) Теперь обратимся к **прогнозированию** на основе АРПСС моделей (и их более сложных вариаций). Загрузите из mat-файла **Fort.mat** массив, содержащий отсчеты некоторого реального ВР, всего 174 отсчета в вектор-строке.

```
file = h5py.File('Fort.mat', 'r')
data = file.get('Fort')
Fort = np.array(data)
Fort = Fort.ravel()
plt.figure(figsize = (10, 5))
plt.plot(Fort, 'k')
plt.show()
```

29) Мы будем производить **ретроспективный прогноз**. Для этого отрежем от данного ряда последние 24 точки (которые мы и будем прогнозировать):

```
F = Fort[:len(Fort)-24+1] # отрезаем последние 24 точки
plt.figure(figsize = (10, 5))
plt.plot(Fort, 'k') # исходный ВР
plt.plot(F, 'b') # урезанный ряд
plt.show()
```

30) Прежде, чем строить модель АРПСС, **обратите внимание**: модели АРПСС строятся для рядов с около-нулевым средним, что неверно для заданного временного ряда. Поэтому – **сначала постройте линейный тренд прогнозируемого ряда** (см. линейную регрессию первого порядка из предыдущей лаб. работы)**, а затем вычтите его из исходного ряда**, приведя его к нулевому среднему значению (к так называемой **тренд-стационарной** форме).

31) У любой подобной распространенной проблемы анализа ВР есть и более изящные решения: оказывается есть готовая функция **detrend** из библиотеки *Python* для решения подобной проблемы:

F_minus_trend = statsmodels.tsa.tsatools.detrend(F)

32) Убедитесь, что Ваши результаты из п. 30 совпадают с результатами из п. 31.

33) Подберите для данного приведенного ВР без тренда, у которого к тому же отрезали последние 24 точки, модель АРПСС (p, d, q) некоторого порядка (все параметры целиком и полностью определяются самим студентом) по таблицам и информационным критериям. Например, была найдена некоторая наилучшая модель:

arimaz = ARIMA(F_minus_trend, order = (*p*, *d*, *q*))

model_fit = arimaz.fit()   **# подгоняем под ВР**

print(model_fit.summary())

34) Тогда итоговые значения модельного ВР и его прогноза по данной модели можно найти очень легко через функцию **predict**, например на длину всего ряда:

model_fit.predict(0, len(Fort))

35) Постройте график этого прогноза **predict** на фоне ВР без тренда **F_minus_trend**.

36) На самом деле можно было обойтись и без убирания тренда из исходных данных, так как современные библиотеки для работы с ВР достаточно продвинуты, чтобы решать подобные проблемы, а если операцию де-трендирования всё-таки необходимо выполнить – обычно об этом выводят отдельный *warning*. Поэтому давайте вернемся к нашему исходному ВР ***Fort***, точнее к ВР, у которого отрезали 24 последние точки для ретроспективного прогноза, то есть к ряду ***F***, и построим некоторую модель ARIMA для него, а затем построим его прогноз. Самостоятельно подберите параметры модели ARIMA для этого ряда (параметры могу отличаться для данных с трендом и без!):

```
arimaz = ARIMA(F, order = (p, d, q))
model_fit = arimaz.fit()        # подгоняем под ВР
print(model_fit.summary())
```

37) Затем постройте его прогноз на фоне исходных данных ***Fort***:

```
plt.figure(figsize = (10, 5))
plt.plot(model_fit.predict(0, len(Fort)), 'k')
plt.plot(Fort, 'g')
plt.show()
```

38) Не забудьте в отчет-тетрадь добавить необходимые рисунки и таблицы результатов. А также свои пояснения по всем полученным результатам.

39) Теперь займемся прогнозом этого ВР другими моделями, которые во многом схожи с моделями ARIMA или даже зачастую используют их как свою важную часть.

40) Сначала используем θ-модель прогноза за авторством Assimakopoulos & Nikolopoulos (2000), основу которой составляет мультипликативная модель экспоненциального сглаживания. **Самостоятельно подберите параметр period** периодичности / сезонности данной модели для получения наилучшего прогноза:

```
from statsmodels.tsa.forecasting.theta import ThetaModel
tm = ThetaModel(F, period=XX)
res = tm.fit()
print(res.summary())
```

41) Для проверки результатов не забывайте строить прогноз получившейся модели в сравнении с исходными данными:

```
plt.figure(figsize = (10, 5))
plt.plot(res.forecast(24), 'k')
plt.plot(Fort, 'g')
plt.show()
```

42) Теперь используем общую мультипликативную сезонную ARIMA модель с дополнительными входами (**SARIMAX**). Эта мультипликативная модель имеет аж 7 (!) параметров порядка без учета входа ***X***: это 3 параметра ARIMA (p, d, q) и 4 параметра сезонности (P, D, Q, S): $(p,d,q)\times(P,D,Q)_s$. А ее общая *краткая* форма записи есть зрелище не для слабонервных:

$$\phi_p(L)\tilde{\phi}_P(L^s)\Delta^d\Delta_s^D y_t = A(t)+\theta_q(L)\tilde{\theta}_Q(L^s)\zeta_t$$

43) Для начала убедимся, что наша модель ARIMA есть частное подмножество модели SARIMAX. Для этого создайте прогноз с помощью модели SARIMAX с параметрами **(*p*, *d*, *q*)** Вашей расчетной модели ARIMA из п. 36 ранее:

```
from statsmodels.tsa.api import SARIMAX
sarimax_mod = SARIMAX(F, order=(p, d, q), trend='ct')
sarimax_res = sarimax_mod.fit()
print(sarimax_res.summary())
plt.figure(figsize = (10, 5))
plt.plot(sarimax_res.predict(0, len(Fort)), 'k')
plt.plot(Fort, 'g')
plt.show()
```

Убедитесь в схожести результатов (хотя бы визуально).

44) А теперь добавим «сезонности» в нашу модель и доведем ее до полного набора в 7 параметров:

```
mod = SARIMAX(F, order=(p, d, q), seasonal_order=(P, D, Q, S), trend='ct')
mres = mod.fit()
print(mres.summary())
plt.figure(figsize = (10, 5))
plt.plot(mres.predict(0, len(Fort)), 'k')
plt.plot(Fort, 'g')
plt.show()
```

45) Самостоятельно **подберите все 7 параметров** для Вашей наилучшей модели SARIMAX. Не забывайте сравнивать модели по информационному критерию **BIC**.

46) *Подсказка*: наиболее важным параметром для модели будет «сезонность» ряда $\boldsymbol{S}$, которую надо определить в первую очередь. А вот параметры $\boldsymbol{q}$ и $\boldsymbol{Q}$ может приравнять 0 и варьировать при необходимости в самую последнюю очередь, так как они вносят наименьший вклад по точности, но при этом являются наиболее вычислительно затратными.

47) Не забудьте в отчет-тетрадь добавить необходимые рисунки и таблицы результатов.

2. Требования к оформлению отчета

Отчет в Jupyter-тетради должен обязательно содержать: номер лабораторной работы, ФИО студента, номер варианта (либо студенческий номер), номер группы, результаты выполнения работы с комментариями студента (комментарии пишутся после #) и изображениями.